

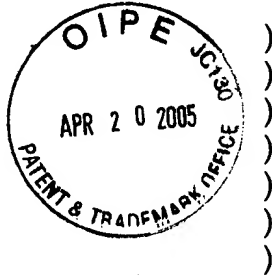
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

### Application of:

HIROSHI IZUMI

**Ser. No.: 09/754,536**

Filed: 1/4/01



COMPUTER AND COMPUTER-  
READABLE STORAGE MEDIUM

**Group Art Unit: 2195**

**Examiner: M. Banankhah**

## TRANSMITTAL OF PRIORITY DOCUMENT AND CLAIM FOR PRIORITY

Commissioner of Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450  
Sir:

Attached hereto is a certified copy of Japanese priority document 2000-042195 filed in Japan on February 21, 2000. Applicant reiterates its claim of priority for this application based on Japanese application 2000-042195.

Respectfully submitted,

By


John S. Mortimer, Reg. No. 30,407

**WOOD, PHILLIPS, KATZ,  
CLARK & MORTIMER**  
500 W. Madison St., Suite 3800  
Chicago, IL 60661  
(312) 876-1800

Date: April 15, 2005

37 CFR 1.8  
CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on 4-18-05 (date).

  
\_\_\_\_\_  
Terri Craine

日 本 国 特 許 庁

PATENT OFFICE  
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されて  
る事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed  
in this Office.

出 願 年 月 日  
Date of Application:

2000年 2月21日

出 願 番 号  
Application Number:

特願2000-042195

出 願 人  
Applicant(s):

和泉 博

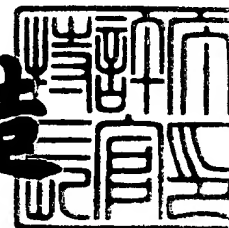
BEST AVAILABLE COPY

CERTIFIED COPY OF  
PRIORITY DOCUMENT

2000年12月22日

特許庁長官  
Commissioner,  
Patent Office

及川耕造



【書類名】 特許願

【整理番号】 P0000670

【あて先】 特許庁長官 殿

【国際特許分類】 G05B 19/00

【発明者】

    【住所又は居所】 長野県木曽郡木祖村小木曽 2 9 7 7

    【氏名】 和泉 博

【特許出願人】

    【住所又は居所】 長野県木曽郡木祖村小木曽 2 9 7 7

    【氏名又は名称】 和泉 博

【代理人】

    【識別番号】 100107593

    【弁理士】

    【氏名又は名称】 村上 太郎

    【電話番号】 06-6356-3476

【手数料の表示】

    【予納台帳番号】 023009

    【納付金額】 21,000円

【提出物件の目録】

    【物件名】 明細書 1

    【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンピュータ、並びに、コンピュータ読み取り可能な記録媒体

【特許請求の範囲】

【請求項 1】 カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして、少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、

前記外部コマンドの一つにファンクションコマンドを備え、該ファンクションコマンドはファイル名を引数として実行可能であり、

ファンクションコマンドがファイル名を引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、前記ファイル名によって示されるファイル内に所定のフォーマットで記述されたスクリプト文字列を入力し、該スクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する、

ことを特徴とするコンピュータ。

【請求項 2】 カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステ

ムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、

前記外部コマンドの一つにファンクションコマンドを備え、該ファンクションコマンドは、所定のフォーマットの関数インデックスを引数として実行可能であり、

ファンクションコマンドが関数インデックスを引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、該プロセスがアクセス可能なメモリ内に予め記憶された関数インデックスの実体を記述するスクリプト文字列を入力し、該スクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する、

ことを特徴とするコンピュータ。

【請求項3】 カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、

前記外部コマンドの一つにファンクションコマンドを備え、該ファンクションコマンドはファイル名若しくは所定のフォーマットの関数インデックスを引数として実行可能であり、

ファンクションコマンドがファイル名を引数として実行されたとき、該ファン

クションコマンドを実行するプロセスが、前記ファイル名によって示されるファイル内に所定のフォーマットで記述されたメインスクリプト文字列を入力し、該メインスクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力し、

ファンクションコマンドが関数インデックスを引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、該プロセスがアクセス可能なメモリ内に予め記憶された関数インデックスの実体を記述する関数スクリプト文字列を入力し、該関数スクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する、

ことを特徴とするコンピュータ。

【請求項4】 ファンクションコマンドの引数として指定されるファイル内に、メインスクリプト文字列及び関数スクリプト文字列を記述可能であり、これらスクリプト文字列内には、関数スクリプト文字列で定義された関数名を引数としてファンクションコマンドのコマンド名を記述可能であり、

ファンクションコマンドがファイル名を引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、プロセス間通信用の共有メモリの確保をカーネルに要求し、該プロセスが、確保された共有メモリ内に、該ファンクションコマンドの引数として指定されたファイル内に記述された前記関数スクリプト文字列を記憶し、

コマンド文字列を新たに起動されるコマンドインタプリタに出力する前に、コマンド文字列中に関数名が記述されていれば、該関数名を、該関数名に関連付けられる関数スクリプト文字列が記憶された前記共有メモリのアドレスを指定する所定のフォーマットの関数インデックスに置換する、

ことを特徴とする請求項3に記載のコンピュータ。

【請求項5】 オペレーティングシステムのコマンドインタプリタにより解釈実行可能なファンクションコマンドのプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、前記ファンクションコマンドは、実行時に少な

くとも一つの引数を指定可能であり、前記プログラムは次のステップを有することを特徴とするコンピュータ読み取り可能な記録媒体。

(1) 引数が、ファイル名であるか若しくは関数インデックスであるかを判別するステップ。

(2) 引数がファイル名であるとき、該ファイル名で示されたファイルから、少なくともメイン関数部を含むとともにサブ関数部を含むことのあるスクリプト文字列を入力し、該スクリプト文字列のメイン関数部をコマンドインタプリタにより解釈可能なコマンド文字列に変換するとともに、スクリプト文字列にサブ関数部が存在すれば該サブ関数部をプロセス間通信用の共有メモリに記憶し、新たにコマンドインタプリタを起動して該コマンドインタプリタに前記コマンド文字列を出力するステップ。

(3) 引数が関数インデックスであるとき、該関数インデックスによって指定される共有メモリ内の前記サブ関数部を読み込み、該サブ関数部をコマンドインタプリタにより解釈可能なコマンド文字列に変換し、新たにコマンドインタプリタを起動して該コマンドインタプリタに前記コマンド文字列を出力するステップ。

【請求項6】 ファンクションコマンドの引数となるファイル名及び関数インデックスには、スクリプト文字列で記述されるプログラムで利用可能な引数を付加的に記述可能であり、ファンクションコマンドは、コマンドインタプリタに受け渡し可能な型の戻り値を有することを特徴とする請求項5に記載のコンピュータ読み取り可能な記録媒体。

【請求項7】 サブ関数部には、スクリプト文字列中に記述可能な関数名を定義することが可能であり、

ファンクションコマンドのプログラムは、コマンド文字列を新たに起動されるコマンドインタプリタに出力する前に、コマンド文字列中に関数名が記述されていれば、該関数名を、該関数名に関連付けられる関数スクリプト文字列が記憶された前記共有メモリのアドレスを指定する所定のフォーマットの関数インデックスに置換するステップを有する、

ことを特徴とする請求項5又は6に記載のコンピュータ読み取り可能な記録媒

体。

【請求項 8】 UNIX系オペレーティングシステムがインストールされたコンピュータにおいて、

手続き型高級プログラミング言語を構成するフロー制御構文、並びに、再帰呼び出し可能な関数呼び出し構文が、ファイルシステムのパスの通ったディレクトリ内に保存されたシェル外部コマンドとして実装されていることを特徴とするコンピュータ。

【請求項 9】 請求項 8 に記載の関数呼び出し構文を構成するファンクションコマンドのプログラムが記録されたコンピュータ読み取り可能な記録媒体。

【請求項 10】 マルチタスクコンピュータシステムにおいて、少なくとも一つの関数定義を有する手続き型高級プログラミング言語で記述されたプログラムが、アクセス可能なファイル内に保存されており、該プログラムを実行するコマンドを有し、該コマンドにより前記プログラムを実行すると、該プログラム内において定義された各関数毎に、該関数の実体を実行するプロセスを生成することを特徴とするコンピュータシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、Linux、BSD、SystemVの他、UNIXの設計思想を取り入れたOS-9等のUNIX系OSや、その他のコマンドインタプリタを包含するマルチタスクOSを搭載するコンピュータ、並びに、該コンピュータで利用可能なファンクションコマンドプログラムが記録された記録媒体に関する。

【0002】

【従来の技術】

UNIX（本明細書においてLinuxを含む）は、最も広く普及しているオペレーティングシステムの一つであり、カーネルと呼ばれる最も基本的な部分の制御を司るプログラムと、シェルと呼ばれる対話型のコマンドインタプリタを包含する。コンピュータに所望の動作をさせるために、キーボードなどの入力端末を利用して所望のコマンドをシェルプロンプトから入力すると、シェルが該コマ

ンドを解釈し、コマンドを実行する。

【0003】

UNIXには、非常に多種多様なコマンドが標準で装備されており、基本コマンドを組み合わせることにより少し複雑な処理を行う新しいコマンドを作り、そのコマンドを更に組み合わせてより高度な処理を行うコマンドを作成することが可能である。このような作業は、シェルスクリプトを用いて従来より広く行われている。

【0004】

シェルスクリプトでは、既存のコマンドを高級プログラミング言語の要素のように使用してプログラムすることができる。シェルスクリプトは、コマンド群が記述されたファイルであって、シェルに対して指示をする一つのプログラムのよう使用される。UNIXのシェルスクリプトでは、繰り返し構文や分岐構文を使うことができ、また、bashシェルではスクリプト内で関数定義を行うことも可能である。したがって、bashシェルでは、C言語等の高級プログラミング言語と比して、実行速度を除き遜色のないシェルプログラミングを行うことができる。したがって、実行速度が問題とならない場合、UNIXでのプログラミングはまずシェルスクリプトで作成し、大きなサイズのデータ処理を行う等の実行速度が要求されるコマンドをC言語でプログラミングしている。

【0005】

【発明が解決しようとする課題】

上記の通り、bashシェルでは関数定義をも行うことができ、簡単に高度なプログラミングを行うことができるが、かかるシェルスクリプトを、ボーンシェル（sh）やCシェル（csh）などの他のシェルで利用することはできない。

【0006】

また、シェルスクリプトの場合、繰り返し構文や分岐構文は、シェルの内部に組み込まれたプログラムにより実現されており、この組み込みプログラムは、シェルスクリプトのプロセスにより実行される。したがって、これらのフロー制御構文によるプログラムの流れから強制的に脱出する（例えば、`^D`をタイプすることにより強制終了する）ことは、同時に、シェルスクリプトプログラムを実行す

るプロセスの強制終了を意味する。ユーザーがシェルスクリプトを利用して作業を行っている場合に、そのシェルスクリプトでプログラムされている一部のループから強制的に抜け出し、その後に記述されているコマンドを継続して実行したいこともある。しかし、従来のシェルプログラミングでは、そのようなケースを予め想定して、ループから抜け出すための流れ制御構造を予めプログラムしておく必要がある。

#### 【0007】

本発明は、関数をシェル外部コマンドとして実現することにより、全く新たな発想のプログラミング環境を提供することを目的とする。また、本発明は、ユーザーが使用しているシェルが付加的に提供する機能によることなく、関数定義を用いた高度なプログラムを作成できるようにし、shシェルやcshシェルなどの伝統的なシェルの改造することなく、C言語と同様のプログラミングを行えるようにすることを目的とする。また、本発明は、オペレーティングシステムがユーザーに提供する機能を余すところなく利用して、柔軟かつ強力なプログラミングを行えるようにすることを目的とする。

#### 【0008】

##### 【課題を解決するための手段】

本発明は、UNIX系オペレーティングシステムがインストールされたコンピュータにおいて、手続き型高級プログラミング言語を構成するフロー制御構文、並びに、再帰呼び出し可能な関数呼び出し構文を、ファイルシステムのパスの通ったディレクトリ内に保存されたシェル外部コマンドとして実装したことを特徴とするものである。これによれば、外部コマンドの組み合わせのみでプログラミングを行うことが可能であり、シェルの内部に組み込まれた流れ制御構文を用いることなく、C言語と同様の高度なプログラミングを行うことが可能である。また、流れ制御構文や関数呼び出し構文が、シェル外部コマンドとして実装されているから、ループや条件分岐が、個々に生成された子プロセスにおいて行われ、ループなどからの強制的な脱出を行っても、そのループを制御する子プロセスが強制終了するのみで、プログラムを実行する基本となるプロセスは生きているため、プログラムされたその後の処理が継続して行われる。

## 【0009】

本発明の構文コマンドを用いたプログラムファイルは、シェルスクリプトとして実行してもよく、また、新たに作成したインタプリタコマンドで実行することも可能である。どのように利用するか of 最終的な選択は、ユーザーに委ねられる。本願発明者が推奨する最も最良の選択は、後述するファンクションコマンドを利用する方法である。このファンクションコマンドは、手続き型高級プログラミング言語を実現する一種のインタプリタとして機能してもよく、また、所定のフォーマットで作成されたプログラムファイル（スクリプト文字列）をシェル（コマンドインタプリタ）が解釈可能な文字列に変換するという意味で、一種のフィルタとして機能してもよい。

## 【0010】

本発明の新たなプログラミング手法は、UNIX系OSに限定されることなく、カーネル及びコマンドインタプリタを具備する種々のマルチタスクOS上で利用することが可能である。即ち、本発明は、カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして、少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、前記外部コマンドの一つにファンクションコマンドを備え、該ファンクションコマンドはファイル名を引数として実行可能であり、ファンクションコマンドがファイル名を引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、前記ファイル名によって示されるファイル内に所定のフォーマットで記述されたスクリプト文字列を入力し、該スクリプト文字列を、コマンドイ

インタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する、ことを特徴とするコンピュータである。かかるファンクションコマンドがインストールされたコンピュータによれば、柔軟かつ強力なマルチタスクプログラミングを容易に行うことができる。

#### 【 0 0 1 1 】

また、本発明は、カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、前記外部コマンドの一つにファンクションコマンドを備え、該ファンクションコマンドは、所定のフォーマットの関数インデックスを引数として実行可能であり、ファンクションコマンドが関数インデックスを引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、該プロセスがアクセス可能なメモリ内に予め記憶された関数インデックスの実体を記述するスクリプト文字列を入力し、該スクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する、ことを特徴とするコンピュータである。これによれば、プログラムの関数の実体を構成するコマンドを起動するプロセスは、プログラムの本体（メイン関数部）を構成するコマンドを起動するプロセスの子プロセスとなるため、関数部の実行中に強制終了を行っても、ファンクションコマンドなどの各種コマンドの組み合わせからなるプログラムの本体を実行するプロセスは生きており、かかるプログラムを

継続して実行することが可能となる。

【 0 0 1 2 】

また、本発明は、マルチタスクコンピュータシステムにおいて、少なくとも一つの関数定義を有する手続き型高級プログラミング言語で記述されたプログラムが、アクセス可能なファイル内に保存されており、該プログラムを実行するコマンドを有し、該コマンドにより前記プログラムを実行すると、該プログラム内において定義された各関数毎に、該関数の実体を実行するプロセスを生成することの特徴とするコンピュータシステムである。このようなプログラミング言語としては、後述する F 言語を採用することができ、その他の適宜の高級言語として実施可能である。また、上記コマンドとしては、例えば、上記したファンクションコマンドを用いることができ、また、コマンドインタプリタ（UNIX シェルなど）を利用せずにプログラムを実行するものとしてもよい。

【 0 0 1 3 】

【発明の実施の形態】

以下、本発明の好ましい実施の形態について説明する。

本発明は、カーネル及びコマンドインタプリタを包含するオペレーティングシステムがインストールされており、カーネルが、少なくともファイルシステム及びメモリを管理するとともにマルチタスク制御を行うものであり、コマンドインタプリタが、一つ若しくは複数のコマンドを含むコマンド文字列を入力して該コマンド文字列を解釈し、該文字列で指定された制御内容にしたがってコマンド文字列に含まれる各コマンドを実行させるものであり、前記コマンドとして少なくとも外部コマンドがあり、外部コマンドのプログラムは、ファイルシステムの所定のディレクトリ内に保存された実行可能ファイル内に記述されており、各プログラムは、各プログラム毎にカーネルが生成するプロセスにより実行され、各プロセスの実行状態がカーネルにより制御され、前記コマンドインタプリタが外部コマンドとして実装されたコンピュータにおいて、前記外部コマンドの一つにファンクションコマンドを備えたものとして実施することができる。このファンクションコマンドはファイル名若しくは所定のフォーマットの関数インデックスを引数として実行可能である。ファンクションコマンドがファイル名を引数として

実行されたとき、該ファンクションコマンドを実行するプロセスが、前記ファイル名によって示されるファイル内に所定のフォーマットで記述されたメインスクリプト文字列を入力し、該メインスクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する。また、ファンクションコマンドが関数インデックスを引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、該プロセスがアクセス可能なメモリ内に予め記憶された関数インデックスの実体を記述する関数スクリプト文字列を入力し、該関数スクリプト文字列を、コマンドインタプリタにより解釈可能なコマンド文字列に変換し、該コマンド文字列を、新たに起動されるコマンドインタプリタに出力する。

## 【 0 0 1 4 】

上記オペレーティングシステムの代表的なものは、UNIXである。しかし、本発明は、UNIX以外の他のオペレーティングシステムに適用することも可能である。

## 【 0 0 1 5 】

カーネルは、モノシリックカーネルであってもよく、マイクロカーネルであってもよい。モノシリックカーネルを採用するOSとしては、マイクロソフト社のWindows98、アップル社のMacOS、マイクロウェア社のOS-9などがあり、マイクロカーネルを採用するOSとしては、Be社のBeOS、マイクロソフト社のWindowsNT、Windows2000、アップル社のNeXTSTEP/OPENSTEP、Rhapsody、及び、MkLinuxなどがある。

## 【 0 0 1 6 】

コマンドインタプリタは、UNIXのシェル、マイクロソフト社のMS-DOSやMS-Windowsのコマンドインタプリタなど、OSに標準で、若しくは追加的にインストールされた適宜のものを採用できる。

## 【 0 0 1 7 】

カーネルが管理するファイルシステムは、単一のコンピュータのハードディスクなどの資源のみから構築されるものであってもよく、また、ネットワークディレクトリがマウントされたものであってよい。殆どのOSでは、階層的ファイル

構造（特に木構造）のファイルシステムが採用されているが、本発明は非階層的ファイル構造のファイルシステムを採用する場合でも実施可能である。

【0018】

また、メモリ管理システムとして、仮想メモリシステムを採用していてもよく、スワッピングによる性能低下を回避するために仮想メモリシステムを採用しないOS-9のようなOSでも本発明は実施可能である。

【0019】

マルチタスクの方式としては、プリエンプティブ・マルチタスク (Preemptive Multitask) と、プログラム側で実行権を別のプログラムに渡す協調的マルチタスク (Non Preemptive Multitask) のいずれを採用してもよい。プリエンプティブ・マルチタスクの制御方式としては、一般に、時分割システムが採用される。また、対称型マルチプロセッシング (SMP) 又は非対称型マルチプロセッシングのいずれのマルチプロセッサ対応OSでも実施可能である。対称型マルチプロセッシングは、複数のCPU全てが対等の関係で動作し、どの処理をどのCPUで行うかをOSが自動的に割り当てるようになっており、割り当て時に最も負荷の少ないCPUに処理が割り当てられる。非対称型マルチプロセッシングは、複数のCPUのうちのどれかをメインCPU、他をサブCPUとして、OSなどの処理はメインCPUに、コマンド（アプリケーション等）の処理をサブCPUに処理を割り当てる仕組みになっている。

【0020】

時分割システムでは、複数のプロセスが起動状態にあっても、ある時点で実際に実行されているのは実行権の割り当てられた一つのプロセスである（特にシングルプロセッサの場合）。

【0021】

UNIXでは、プロセスはforkシステムコールによって生成され、メモリ資源やスワップ領域資源が確保されるとそのプロセスは実行可能状態 (runnable, ready) となり、入出力を行うとプロセスは待ち状態 (sleep, blocked, waiting) に状態遷移し、入出力を完了するとまた実行可能状態に戻る。プロセスはSTOPシグナルを受け付けるか、ptraceシステムコールを発行すると停止状態 (stop) にな

り、STOPシグナルを受けて停止状態になったプロセスはCONTシグナルを送ると元の実行可能状態に戻り、ptraceによって停止したプロセスはptraceによって元に戻る。また、プロセスは、exitシステムコールを発行することで自ら終了するとともに、強制的にプロセスを終了させる場合には、該プロセスにkillシグナルを送る。このようにして終了した（死んだ）プロセスはゾンビ状態(zombie)となり、ゾンビ状態のプロセスは、親プロセスがwaitシステムコールを発行することで完全に消滅する。また、親プロセスが子プロセスよりも先に死んだ場合には、カーネルがその子プロセスを他のプロセス（通常は、プロセスIDが1のプロセス）の養子にする（つまり、その子プロセスの持つ属性を書き換える）。実行可能状態、停止状態、待ち状態を合わせてプロセスの活動状態（active）という。

上記実行可能状態は、プロセスにCPU資源が割り当てられれば実行することができる状態である。マルチタスクシステムでは、複数のプロセスが実行可能状態にあることが通常であるが、一般に、優先順位が最も高い一つのプロセスだけが実行中(running)となる。実行中以外の実行可能なプロセスは、実行待ち行列(run queue)に繋がれており、順次CPU資源が割り当てられる。

#### 【 0 0 2 2 】

UNIXにおけるプロセス制御の一例を説明したが、本発明は、これと同一のプロセス制御を行うOSに限定されることなく、種々のプロセス制御方法を採用するOSで実施することが可能である。

#### 【 0 0 2 3 】

上記コマンド文字列の書式はどのようなものでもよく、例えばUNIXシェルの場合、単一のコマンド名及びその引数からなる文字列、若しくは、複数のコマンドを組み合わせてなる文字列などがコマンド文字列となる。コマンド文字列中には、コマンド名、コマンドの引数名、コマンドオプション指定子や、シェルのメタキャラクタを含むことができる。UNIXシェルでは、このメタキャラクタによって、ユーザーが各コマンドの実行方法の制御を指定したり、各コマンドを実行するプロセス間の入出力制御を指定することが可能である。

#### 【 0 0 2 4 】

このようなメタキャラクタとしては、下記のことを挙げることができる。

リダイレクション      :   < > >>  
 パイプ                 :   |  
 コマンド置換           :   ` `   
 シーケンシャルコマンド実行   :   ;  
 バックグラウンドプロセス    :   &  
 コマンドのグループ化        :   (  
 ワイルドカードの置き換え     :   \* ? [  
 変数指定文字            :   \$

## 【0025】

また、UNIXシェルでは、コマンド間を区切るための内部フィールド分離文字(internal field separator) がシェル変数IFSで定義されており、通常これらの文字として、空白、タブ文字、改行文字が定義されている。上記コマンド文字列は、かかる分離文字を含むことができることは勿論である。

その他、使用するコマンドインタプリタ内で予め定められた書式のコマンド文字列を用いることが可能である。

## 【0026】

また、UNIXシェル（例えば、csh, tcsh, bashなど）でプロセスを作ると、同時にジョブも生成される。上記のパイプでつながったプロセス群は、1つのジョブを構成する。1つのジョブを構成するプロセスは同時に扱うことが可能であり、このジョブの実行中にCTRL-Zを入力すると、このジョブを構成する複数のプロセスを同時に停止させることができる。本発明は、このような複数のプロセスにより構成されるジョブのジョブ制御を行いうるOSにおいて好適に実施できる。

## 【0027】

本発明のファンクションコマンドの実体は、好ましくは、C言語などのコンパイラにより作成されたバイナリファイルとしてファイルシステムに保存しておくのがよいが、CPUの処理能力が充分大きいならばシェルスクリプトなどのインタプリタで処理可能なテキストファイルとして保存されたものでもよい。また、少なくとも、オペレーティングシステムの起動時にファイルシステム中に保存

されていればよい。例えば、仮想ディスクシステムを採用するOSの場合には、現実には実メモリ若しくは仮想メモリ内にファンクションコマンドが取り込まれていることもあるが、コマンドインタプリタが外部コマンドとして扱うことのできるものであればよく、ファンクションコマンドの実体が外部記憶装置上にあるかメモリ上にあるかは問題ではない。

## 【0028】

ファンクションコマンドは、ファイル名、関数インデックスの他、種々の起動オプションを引数とすることも可能である。ファイル名を引数としてファンクションコマンドを実行すると、該コマンドがファイル実行モードで起動され、関数インデックスを引数として起動すると関数モードで起動されるように、ファンクションコマンドをプログラムしておく。また、オプションが指定されたときには、このオプション毎に予め定められた動作若しくは処理を行うようにプログラムしておく。

## 【0029】

引数となる関数インデックスは、意味を表す関数名とすることもできるが、関数の実体が記憶されているメモリへの参照を容易に行えるように、そのメモリの番地を指定する16進数表記のアドレス指定文字列を含むものとするのが良い。アドレス指定文字を含む関数インデックスを採用する場合、メモリへの不正なアクセスを行うハッカー行為を防止するため、キーボードなどの入出力端末からは入力することが不可能な文字を関数インデックスの一部に含むものとすることができる。このような入力不可文字としては、キャンセルコード(0x18)などを挙げることができる。例えば、関数インデックスの書式としては、"%...FFFFFF()"とすることができる。ここで、'%'は関数であることを明示するものであり、ファンクションコマンドを関数モードで起動することを識別するとともに、C言語プログラムとの混同を避けるために有用である。また、'.'はキャンセルコード(CAN)である。また、'FFFFFF'がアドレス指定文字列であり、最後の'()'内には、該関数への引数を記述することが可能である。

## 【0030】

ファンクションコマンドの引数として指定されるファイル内には、スクリプト

文字列からなるプログラムを記述しておく。このスクリプト文字列は、ファンクションコマンドによって所定の処理を行ってコマンドインタプリタに出力されるコマンド文字列の元になるプログラムである。このプログラムの書式は適宜のものとすることができ、例えば、C言語と類似の書式とすることができる。ファンクションコマンドは、プリプロセッサ処理を行うものとすることもでき、これによれば一層C言語との類似性が高まり、C言語に慣れたユーザーが容易にファンクションコマンドを利用したプログラムを作成することが可能になる。また、ファンクションコマンドは、広域変数、局所変数を扱えるように構成することも可能である。ファンクションコマンドは、一種のフィルタとして機能し、上記スクリプト文字列から、一定の規則にしたがって、コマンドインタプリタが解釈可能な上記のようなコマンド文字列を生成する。

#### 【0031】

上記スクリプト文字列からなるプログラムは、表面上C言語と極めて類似の書式のものとすることができるが、その実行形態はC言語やシェルスクリプトとは大きく異なる。C言語は、実行前にコンパイル作業が必要であるが、本発明のファンクションコマンドを利用したプログラムはコンパイルの必要がない。また、シェルスクリプトは、そもそもシェルが直接解釈可能なコマンド文字列の集合であるが、本発明におけるスクリプト文字列は、ファンクションコマンドのプログラミングによって種々の書式とすることができ、上記のようなC言語と酷似の言語を新たに提供することもでき、また、PascalやFortranに類似の言語を新たに提供することもできるだろう。さらに、シェルスクリプトとの根本的な相違点は、個々の関数毎に、新たなプロセスが生成される点である。かかる特徴によって、シェルスクリプトよりも柔軟にマルチタスクプログラミングを行うことが可能になる。

#### 【0032】

関数インデックスの実体を記述する関数スクリプト文字列は、該関数インデックスを引数とするファンクションコマンドが起動される前に、予め所定のメモリ内に格納される。例えば、OS-9では、仮想メモリシステムを採用せず、各プロセスは任意の物理メモリ空間にアクセス可能であるため、ファンクションコマ

ンドを起動するプロセスが、上記関数スクリプト文字列を格納するためのメモリを確保し、該メモリに該文字列を格納した後、ファンクションコマンドを起動すればよい。一方、UNIXなどの、当該コマンドを実行するプロセスがアクセス可能なメモリは、起動時若しくは起動後に割り当てられ、他のプロセスに割り当てられたメモリにアクセスすることはできないようになっている。したがって、当該プロセスの起動前に、他のプロセスによって所定のメモリ内に格納された文字列を読み出すことは通常の方法では不可能である。しかし、マルチタスクOSでは、何らかのプロセス間通信手法が実装されており、適当なプロセス間通信手法を用いて、関数インデックスを引数として起動されたファンクションコマンドに関数スクリプト文字列を与えることが可能である。このような手法の一つに、共有メモリを挙げることができる。

#### 【0033】

スクリプト文字列は、そのプログラムの本体となるメインスクリプト文字列と、プログラム内で使用される関数を定義する関数スクリプト文字列とを、別ファイル内に記述することができるが、単一のファイル内に一連に記述することも可能である。また、プリプロセッサとして#includeを導入することにより、初期定義や関数定義を含む各種定義を別ファイルに記述することもできる。

#### 【0034】

好ましくは本発明のコンピュータは、ファンクションコマンドの引数として指定されるファイル内に、メインスクリプト文字列及び関数スクリプト文字列を記述可能であり、これらスクリプト文字列内には、関数スクリプト文字列で定義された関数名を引数としてファンクションコマンドのコマンド名を記述可能とする。そして、ファンクションコマンドがファイル名を引数として実行されたとき、該ファンクションコマンドを実行するプロセスが、プロセス間通信用の共有メモリの確保をカーネルに要求し、該プロセスが、確保された共有メモリ内に、該ファンクションコマンドの引数として指定されたファイル内に記述された前記関数スクリプト文字列を記憶する。また、コマンド文字列を新たに起動されるコマンドインタプリタに出力する前に、コマンド文字列中に関数名が記述されていれば、該関数名を、該関数名に関連付けられる関数スクリプト文字列が記憶された前

記共有メモリのアドレスを指定する所定のフォーマットの関数インデックスに置換するように構成できる。これらの構成は、主として、ファンクションコマンドにこれらの機能を実装する（このような機能を達成するようにプログラムすることによって得ることが可能である。

#### 【0035】

「コマンド文字列を新たに起動されるコマンドインタプリタに出力する前」とは、当該出力を行うプロセスが起動される前でもよく、この場合には、当該出力を行うプロセスよりも上位のプロセス（例えば親プロセス）が関数名から関数インデックスへの変換を行い、共有メモリに関数スクリプト文字列を記憶すればよい。また、当該出力を行う直前に当該出力を行うプロセス（つまり、関数インデックスを引数として起動されたファンクションコマンド）が前記変換を行ってもよい（この場合、共有メモリに記憶されている関数スクリプト文字列には、関数インデックスではなく、関数名が記述されることになる）。

#### 【0036】

本発明は、コンピュータとして実施できるのみならず、上記コンピュータを構成するためのファンクションコマンドのプログラムが記録された記録媒体としても好適に実施できる。また、当該ファンクションコマンドのプログラムを、インターネット等の通信回線を通じて提供するためのサーバー或いは伝送媒体として好適に実施できる。

#### 【0037】

より詳細には、本発明は、オペレーティングシステムのコマンドインタプリタにより解釈実行可能なファンクションコマンドのプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、前記ファンクションコマンドは、実行時に少なくとも一つの引数を指定可能であり、前記プログラムは次のステップ

（１） 引数が、ファイル名であるか若しくは関数インデックスであるかを判別するステップ

（２） 引数がファイル名であるとき、該ファイル名で示されたファイルから、少なくともメイン関数部を含むとともにサブ関数部を含むことのあるスクリプト文字列を入力し、該スクリプト文字列のメイン関数部をコマンドインタプリタに

より解釈可能なコマンド文字列に変換するとともに、スクリプト文字列にサブ関数部が存在すれば該サブ関数部をプロセス間通信用の共有メモリに記憶し、新たにコマンドインタプリタを起動して該コマンドインタプリタに前記コマンド文字列を出力するステップ

(3) 引数が関数インデックスであるとき、該関数インデックスによって指定される共有メモリ内の前記サブ関数部を読み込み、該サブ関数部をコマンドインタプリタにより解釈可能なコマンド文字列に変換し、新たにコマンドインタプリタを起動して該コマンドインタプリタに前記コマンド文字列を出力するステップ

を有することを特徴とするコンピュータ読み取り可能な記録媒体として好適に実施できる。このような記録媒体としては、CD-ROM、CD-R、CD-RW、フロッピーディスク、ハードディスク、MOディスク、Zipディスク、DVD-ROM、DVD-RAM、スーパーディスクなどの任意のものであってよい。なお、「メイン関数部」は、上記本発明のコンピュータにおけるメインスクリプト文字列に対応し、「サブ関数部」は関数スクリプト文字列に対応する。

#### 【0038】

上記ファンクションコマンドのプログラムは、例えばC言語やC++などのコンパイラで作成することが好ましい。また、該プログラムは、ソースリストの形で記録してもよく、また、コンパイル後のバイナリ形式で記録してもよく、redhat系Linux向けに配布する場合にはrpm形式で記録してもよく、debian向けに配布する場合にはdeb形式で記録してもよい。

#### 【0039】

さらに、上記本発明の記録媒体は、ファンクションコマンドの引数となるファイル名及び関数インデックスには、スクリプト文字列で記述されるプログラムで利用可能な引数を付加的に記述可能であり、ファンクションコマンドは、コマンドインタプリタに受け渡し可能な型の戻り値を有することを特徴とするものとして実施することも可能である。

#### 【0040】

また、上記記録媒体において、サブ関数部には、スクリプト文字列中に記述可能な関数名を定義することが可能であり、ファンクションコマンドのプログラム

は、コマンド文字列を新たに起動されるコマンドインタプリタに出力する前に、コマンド文字列中に関数名が記述されていれば、該関数名を、該関数名に関連付けられる関数スクリプト文字列が記憶された前記共有メモリのアドレスを指定する所定のフォーマットの関数インデックスに置換するステップを有する、ものとしてもよい。

#### 【0041】

また、本発明は、UNIX系オペレーティングシステムがインストールされたコンピュータにおいて、手続き型高級プログラミング言語を構成するフロー制御構文、並びに、再帰呼び出し可能な関数呼び出し構文が、ファイルシステムのパスの通ったディレクトリ内に保存されたシェル外部コマンドとして実装されているコンピュータとして実施でき、このコンピュータにインストールされる関数呼び出し構文を構成するファンクションコマンドのプログラムが記録されたコンピュータ読み取り可能な記録媒体として実施することも可能である。

#### 【0042】

上記フロー制御構文としては、条件分岐構文（if構文、case構文など）や繰り返し構文（while構文、for構文、until構文など）を例示することができる。

例えば、条件分岐構文となるifコマンドとして、評価式（若しくは評価値）を与える文字列と、少なくとも評価式が真であるときに実行すべきコマンド文字列とを引数として起動し得るようにプログラムされたものを用いることができる。かかるifコマンドの一例を挙げれば、次の通りである。

#### 【0043】

ifコマンド

〔書式〕

```
if [option] 制御コマンド1 コマンド群1
    [else if [option] 制御コマンド2 コマンド群2]
    [else コマンド群3]
```

ここで [] 内は省略可能な引数である。

#### 【0044】

このifコマンドを起動すると、まず、制御コマンド1を実行し、そのコマンド

の戻り値を評価値として、これが真(True)ならコマンド群1を実行する。また、制御コマンド1の戻り値が偽(False)なら、制御コマンド2を実行し、その戻り値が真ならコマンド群2を実行し、偽ならコマンド群3を実行する。

このifコマンドは、シェルスクリプトで利用可能なif構文とは異なり、新たに生成されたプロセスで実行されるものであるから、ifコマンドの引数となる制御コマンドやコマンド群を強制終了しても、ifコマンドを起動した親プロセスは実行を継続し、ifコマンド以降に記述されたコマンドの実行を継続できる。

#### 【0045】

上記関数呼び出し構文をシェル外部コマンド（以下、funcコマンドと称する）として実装する方法としては、個々の関数毎にスクリプトファイルを作成しておき、これをfuncコマンドの引数として起動してもよく、また、単一のスクリプトファイル内にメイン関数部とサブ関数部を記述しておき、これを引数としてfuncコマンドを起動すると、関数呼び出し毎に新たなプロセスを生成して関数部を解釈してシェルにコマンド文字列を出力するようにしてもよい。

#### 【0046】

なお、上記した本発明において、各関数を実行するプロセスは、自己が実行すべきスクリプト文字列を記憶する共有メモリを新たに確保し、該メモリに記憶したスクリプト文字列を読み出してコマンド文字列に変換するように構成することが好ましい。

#### 【0047】

#### 【実施例】

本願発明者は、以下に述べる機能を有するファンクションコマンド(func)をC言語を用いて作成した。このfuncコマンドによって実現される新たな言語を「F言語」と称し、以下、このF言語の詳細を説明する。なお、以下の作業は、OSとしてLinuxを用い、コマンドインタプリタとしてBashを用いて行った。また、ファンクションコマンドのファイル名をfuncとし、該funcコマンドはファイルシステムのパスの通ったディレクトリ（例えば、/usr/binや/binなど）内に保存されているものとする。本実施例のfuncコマンドは、以下に説明するように動作するものであり、その動作内容から当業者であればC言語等を用いて容易にプ

ログラミングすることができるであろう。また、以下の説明から、当業者であれば本発明のコンピュータ並びに記録媒体を容易に実施可能となるであろう。

## 【0048】

まず、“hello world”を印字するF言語プログラムを紹介する。このプログラムは、本発明におけるスクリプト文字列として、funcコマンドによって処理され、コマンドシェルによって解釈実行される。

## 【0049】

```
%main($void)
{
    echo "hello, world!"
}
```

## 【0050】

F言語は、C言語のようなコンパイラではないから、コンパイルやリンクの作業は不要である。上記したプログラムをviやemacsなどの適当なエディタで入力し、適当なファイル名で保存し、そのファイル名を引数としてファンクションコマンドをシェルから実行すれば良い。ファイル名を test とすれば、

```
func 'test()'
```

とシェルプロンプトからキー入力し、リターンキーを押すことで上記プログラムを実行できる。引数を'で括ったのは、Bashシェルがファイル名に続く括弧文字()を、コマンドグループを示すメタキャラクタと解釈し、文法エラーとなるためである。上記のようにファイル名と括弧文字を含む文字列全体を'で括ることにより、該文字列 test() を引数としてfuncコマンドが起動される。なお、シングルクォーテーションに代えて、ダブルクォーテーション"で引数を括っても同様の結果が得られる。なお、OS-9のシェルの場合には、括弧文字がメタキャラクタとして定義されていないため、'で括ることなく test() という文字列を引数として与えることが可能である。また、上記の例では、引数が存在しないから、func test とファイル名のみを引数として実行することも可能になっている。

## 【0051】

上記プログラム1を実行すると、funcコマンドは、上記プログラムファイルからスクリプト文字列を読み込み、該文字列を、

```
echo "hello, world!"
```

というコマンド文字列に変換し、子プロセスとして新たなシェルを起動して、該シェルにコマンド文字列を出力する。すると、子シェルは、このコマンド文字列を解釈し、echoコマンドを起動して、端末に"hello, world!"と出力する。

#### 【0052】

F言語のプログラムは、%main()関数から始めることとする。頭の%は、C言語とプログラム書式が酷似しているため、C言語との混乱を回避するためのものである。続く()内にはパラメータが記述され、このパラメータの先頭には\$を記述するものとする。このパラメータには、funcコマンドの起動時に指定された引数が代入される。なお、Fプログラムに受け渡すべき引数がない場合でも()内を空文字とすること(省略すること)はできない。Fプログラムに受け渡すべき引数がない場合には、上記の例のように(\$void)と明記するものとする。既に説明したように、関数は個々に独立しているから、広域変数と重ならない限り自由なパラメータ名を用いることができる。

#### 【0053】

F言語には、C言語と同様のプリプロセッサ機能が具備されている。ライブラリファイル myfile.h に

```
#define printf echo
```

と記述されていれば、上記testプログラムは、C言語に似せて次のようにも記述することができる。なお、この例のように、C言語と同様の注釈を用いることも可能である。

#### 【0054】

```
#include <myfile.h> /* ライブラリの情報を取り込む*/
```

```
%main($void) {                                // 引数のない%main関数を定義
    printf "hello, world!" // コマンドを実行、印字する
}
```

【0055】

次に、最大公約数を求めるFプログラムを紹介する。

【0056】

```
/* Greatest common divisor By Bash.

    file name: gcd
    execution: func "gcd(a, b, NULL)"
                a, b : some integer number
    example: func "gcd(128, 72, NULL)"

*/

#include <myfile.h>

%main($a, $b, $call) {
    return '$b == 0' ? 'echo " G.C.D= $a ($call times)"' ¥
        : 'func "%main($b, $[$a % $b], $[$call + 1])"' ;
}
```

【0057】

一見すると、恰も三項演算子の機能があるかのように見えるが、既に述べたように、F言語はコマンドの集合体であるから、そのようなコマンドが用意されていなければ実現できない。この例では、三項演算子の機能は、myfile.h に次の定義を行い、if分岐コマンドで代用することにより実現している。

```
#define 'return X ? Y : Z' "if '(X)' '(Y)' else '(Z)'"
```

この#defineプリプロセッサにより、return構文はifコマンドに置換され、大文字の部分が対応する文字列に置き換わる。F言語は小文字を原則とするが、#defineプリプロセッサにおいては、大文字は変数として扱われる。

【0058】

上記最大公約数を求めるFプログラムをfuncコマンドで実行すると、該funcコ

マンドのプロセスは共有メモリの確保をカーネルに要求する。カーネルは、指定されたサイズの共有メモリを確保し、funcコマンドのプロセスは、Fプログラムを確保された共有メモリに保存する。そして、上記プログラム（スクリプト文字列）は、funcコマンドによって、次のようなコマンド文字列に変換され、子シェルに出力される。

【0059】

```
if '($b == 0)' '(echo " G.C.D= $a ($$call times)")' else '(func "%main($b, $a % $b), $$[$call + 1] )"' ;
```

【0060】

子シェルは、このコマンド文字列を解釈し、条件分岐構文を構成する一実施例に係る外部コマンドの一つであるifコマンドを実行する。このifコマンドは、パラメータ\$bが0であるか否かを評価し、真であればパラメータ\$aを最大公約数として表示し、偽であれば、再帰的に%main関数を実行するfuncコマンドを子プロセスとして起動する。このfuncコマンドのプロセスは、共有メモリ内に既に記憶されているスクリプト文字列から%main関数を探索し、該当文字列を読み込んで上記処理を繰り返す。なお、シェルに出力されるコマンド文字列中に" func %関数" が記述されている場合、その関数の実体が記憶されている共有メモリのアドレスを指定する文字列に変換した後、シェルに出力することも可能である。即ち、上記の例の場合、子シェルに出力されるコマンド文字列は、次のようなものとする事が可能である。

【0061】

```
if '($b == 0)' '(echo " G.C.D= $a ($$call times)")' else '(func "%shraddress($b, $a % $b), $$[$call + 1] )"' ;
```

【0062】

ここで、shraddressは、%main関数が記憶された共有メモリのアドレスであり、16進数表記のものとすることが可能である。このように、予め関数名を、アドレス指定を含む関数インデックスに置換しておくことにより、関数を実行するために子プロセスとして起動されたfuncコマンドが、共有メモリ内から関数の実体を探索することが必要なくなり、システムのオーバーヘッドを低減できる。

## 【0063】

次に、F言語における関数定義について説明する。F言語の関数はC言語と同様であるが、大きな違いは、戻り値(return value)の型(type)が、シェルが読み込みできるバイト列(文字列)の一つしかないことである。関数の一般形は、次の通りである。

## 【0064】

```
%function-name(parameter declarations, if any)
{
    declarations
    statements
}
```

## 【0065】

この関数定義の最初の行には、その関数名と、パラメーターとが定義され、{ } 内には、関数の実体がスクリプト文字列として定義される。declarationsには、変数の宣言などを行う。statementsには、コマンドシェルが解釈可能なコマンドや、シェルで定義されているメタキャラクタ、プリプロセッサで定義された文字列などを記述可能である。

関数の例を示すものとして、べき乗を求める関数%powerと、それをテストする主関数%mainを示す。なお、このFプログラムのファイル名を power とする。

## 【0066】

```
/* Power program by Bash.
   file name: power
   execution: func "power(m, n)"
*/

%main($m, $n)
{
    echo -n " power($m, $n) = ";
    func "%power($m, 1, $n)";
}
```

}

`%power($m, $a, $n)`

{

`if '($n == 0)' '{``echo $a;``}' else '{``func "%power($m, $[$m * $a], $[$n - 1])"`

}

【0067】

ここで、`%main()`の下での`{ }`で囲まれた部分が、メインスクリプト文字列（スクリプト文字列のメイン関数部）であり、`%power()`の下での`{ }`で囲まれた部分が、関数スクリプト文字列（スクリプト文字列のサブ関数部）である。

`%power()`内で使用されるパラメータは、この関数内で用いられる局所的なものであり、C言語と同様に他の関数からアクセスすることはできない。したがって、他の関数で用いられている変数名と矛盾することなく同じパラメータ名を用いることができる。`%power()`で計算した値は、`echo`コマンドなどを用いることによって`%main()`に返すことができる。勿論、戻り値のない関数を定義することもできる。

上記Fプログラムを実行すると、`func`コマンドプロセスは、確保された共有メモリに、このプログラム（スクリプト文字列）を、シェルにより解釈実行可能な形式に変換して記憶する。この共有メモリに記憶される文字列は、次のようなものである。ここに示したように、関数の実体を括る`{ }`は`( )`に置換され、各関数内の改行コードも削除され、関数の実体は、シェルに受け渡し可能な一連のコマンド文字列に変換されている。

【0068】

`%main($m, $n)``echo -n " power($m, $n) = "; func "%power($m, 1, $n)";`

```
%power($m, $a, $n)
if '($n == 0)' '(echo $a;)' else '(func "%power($m, $[$m * $a], $[$n -
1])")'
```

【 0 0 6 9 】

powerファイルを引数として起動されたfuncコマンドは、子プロセスとして起動したシェルに、%main関数部の実体部分をコマンド文字列として出力する。子シェルは、echoコマンドを実行するとともに、%power関数への関数インデックスを引数としてfuncコマンドを更に子プロセスとして再帰的に起動する。かかる関数インデックスは、関数名であってもよく、%power関数が記憶されている共有メモリへのポインターであってもよい。

【 0 0 7 0 】

F言語の変数としては、広域変数(global variables)と局所変数(local variables)のいずれを用いることもできる。広域変数はプログラム全体のすべての関数に共通の変数である。したがって、広域変数は%main()が動いている間は存在し続けるから、関数間の共通のデータのやり取りに使うことができる。一方、局所変数は、その関数の中でのみ有効な変数のことである。この局所変数は関数毎に独立しているため、同じ名前の変数名を複数の関数で独立して使用することが可能である。広域変数の宣言は、%main()の前で行うものとする。また、局所変数についても、必ず各関数の実体を記述するスクリプト文字列の最初に宣言するものとする。何故ならば、プログラムの途中に記述された変数宣言は、プログラム内のコマンドと見なされ、シェルがエラーを返してくるからである。変数宣言の書式は、どのようなものでも良いが、例えば次のようなものとすることができる。

【 0 0 7 1 】

```
shared $const;           //広域変数の宣言
%main($void)
{
    shared $a, $b;        //局所変数の宣言
    ....
```

## 【 0 0 7 2 】

変数となるデータの型は、バイト列型とポインター型を用いて良い。これらは共に \$ から始まる変数名で表すことができる。バイト列型の変数は、C 言語のように関数内で自由にその値を変えることは許されない。何故なら、F 言語はコマンドの集合体であるとともに関数も個々に独立したプロセスであり、関数の実体となるコマンド文字列をシェルに出力する内部処理構造を採用するため、変数を戻すことができないからである。したがって、関数のパラメータの宣言値は、実行前にその関数内において一律のバイト列として処理される。その性格から、引数のみの宣言に限定されている。

C 言語のように関数内で自由に値を変えることのできるデータ型を提供するために、本実施例の func コマンドでは上記のポインター型を提供している。このポインター型変数は、共有メモリ内に記憶領域を確保することにより、F 言語においても、その関数内で自由に値を変更可能なデータ型を提供する。このポインター型の変数名は、その実現方法に由来する名前をとって、shared で始まるものとするができる。広域変数は、関数を構成するコマンド群間のプロセス間通信に利用できる最高速の伝達媒体として利用できる。局所変数は、各関数毎に、該関数を実行するプロセスに新たに記憶領域が確保され、独立性が保たれる。

## 【 0 0 7 3 】

F 言語は、個々の関数が独立しているため、関数の再帰的な利用が可能である。F 言語において、関数を再帰的に呼び出したとき、即ち、func %関数() を実行したとき、この func コマンドによって新たな共有メモリが確保され、該メモリに該 func コマンドによって処理されるべきプログラムである関数スクリプト文字列を、親プロセスによって確保されている共有メモリ内からコピーする。そして、この func コマンドは、そのコピーした E プログラムを実行する。即ち、%関数() を実行するプロセスは、親プロセスとは完全に独立して実行される。かかる特徴的な内部処理構造に起因して、再帰利用可能な関数の利用価値は非常に大きなものとなる。即ち、マルチタスク OS の特徴である並列処理を、F 言語を用いることで最大限に引き出し、コンピュータ環境を大きく改善することができる。F

言語は、コマンドの集合体であるから、Fプログラム中に、並列処理を行わせるための & を用いることができ、また、%main()を再帰的に呼び出すことも可能である。コマンドを並列起動すれば、上記したように各プロセスは完全に独立しているから、親プロセスであっても自由に強制終了させることができる。このように、F言語では、従来のシェルプログラミングとは比較にならない程、柔軟な並列処理プログラミングを行うことができる。このような処理をC言語で行うには、非常に高度なテクニックと発想の転換とが必要であるが、F言語では、&の設定のみでその処理を簡単に行うことができる。このことから、コンピュータの操作環境を整備するためのプログラムの開発時間を極端に短縮できる。

## 【0074】

このような利点を示すために、一つのプログラム例を紹介する。なお、open\_windowコマンドは、新たにウィンドウを開き、そのウィンドウ内でサブシェルを実行する外部コマンドである。また、whileコマンドは、第1引数として制御コマンドを有し、該制御コマンドの戻り値が真である限り、第2引数として記述されたコマンド群を実行する外部コマンドである。

## 【0075】

```
#include <myfile.h>

%main($directory) {
    open_window {
        while (func %menu($directory)) {
            ...
            if -1 ($1 == "drwx") {
                func %main($9);
            } else {
                more $9 & sleep 10;
            }
        }
        echo "End of file serching.";
    }
}
```

```

        pause;
    } &
}

%menu($direc) {
    echo "Directory: $direc";
    echo;
    ls -l | printf ("%s %s", #1, #9);
}

```

## 【0076】

このFプログラムは、ls -l によってファイルパーミッションとその名前を取り込み、その結果をopen\_windowコマンドで開かれたウィンドウ内においてmoreコマンドで閲覧するものである。そして、表示された中にディレクトリがあれば、%main()を再帰的に呼び出すことによって表示すべきディレクトリがなくなるまでウィンドウを開く。したがって、画面は不要なウィンドウでいっぱいになる。しかし、各ウィンドウは、&を用いて並列起動されているから、親プロセスである重なった不要なウィンドウを自由に消すことができる。

## 【0077】

なお、再帰的に関数を呼び出す場合に、funcコマンド自体に&を用いてはならない。何故なら、funcコマンドを起動しただけで親プロセスに実行が移ってしまい、funcコマンドの引数として指定された関数の実体の実行されないからである。

## 【0078】

以上、F言語の基本構造を説明したが、より一層高度な機能を具備させることも勿論可能である。F言語を快適に動作させるためには、シェルは、コマンドインタプリタに徹することが好ましい。実用的なFプログラムを実行すると、一人あたり100個のプロセスが起動されることが予想される。したがって、Fプログラムでは、極力、余計なメモリ資源を消費しないようにすることが好ましい。また、流れ制御構造もコマンドからなるため、実行結果の終了ステータスは、そ

のまま親プロセスへ返すようにする。

【 0 0 7 9 】

この F 言語によるプログラムの実現方法や、プログラムの設計法は、従来のシェルスクリプトとは全く違うものとなる。プログラムの為に必要な変数があり、その変数は型を有し、さらに、プログラム全体の構成は C 言語と同様の関数の集合体からなっている。F 言語は、各関数が独立していて、局所変数を持っているが故に、再帰的プログラミングが実現できる。この F 言語は、低級な数値計算から、システム制御までをその適応範囲とするインタプリタ言語である。

【 0 0 8 0 】

本発明は、上記実施例に限定されるものではなく、ファンクションコマンドや制御構文コマンドを適宜設計し、提供することができるものである。

【発明の効果】

本発明によれば、従来にない新たな実行形態の再帰的プログラミング可能なインタプリタ型言語を提供でき、ユーザーが自由な発想で自在に自己の操作環境を整備することが可能となる。また、本発明により実現される新たな言語は、そのすべてがシェル外部コマンドにより実現できるから、ある機能を追加する場合は、その機能を有する新たな外部コマンドを作るだけでよく、シェル自体、インタプリタやコンパイラそのものを改良する必要がなく、余分な開発労力を軽減して、開発期間の短縮を図ることができる。したがって、特にエンドユーザーが自己のコンピュータ環境の整備を行う際に好適に使用できるものであるとともに、開発者にとっても、開発効率の向上が図られる。また、統合的な GUI を提供するビジュアルシェルを一括して作成するのではなく、フロントエンドの必要な部分のみを作成すれば良く、結果的に、ユーザーが環境の内部を自由に見ることのできる透過的な環境を構築することが可能である。

【書類名】 要約書

【要約】

【課題】 従来にない新たな実行形態の再帰的プログラミング可能なインタープリタ型言語を提供でき、ユーザーが自由な発想で自在に自己の操作環境を整備することができるようにする。

【解決手段】 UNIXなどのマルチタスクオペレーティングシステムがインストールされたコンピュータにおいて、手続き型高級プログラミング言語を構成するフロー制御構文、並びに、再帰呼び出し可能な関数呼び出し構文を、ファイルシステムのパスの通ったディレクトリ内に保存されたシェル外部コマンドとして実装する。そして、該プログラミング言語で記述されたプログラムは、各関数毎に独立したプロセスで実行されるものとする。

認定・付加情報

特許出願の番号	特願2000-042195
受付番号	50000191564
書類名	特許願
担当官	鈴木 ふさゑ 1608
作成日	平成12年 4月 7日

<認定情報・付加情報>

【提出日】	平成12年 2月21日
-------	-------------

出 願 人 履 歴 情 報

識別番号 [500079403]

1. 変更年月日	2000年 2月21日
[変更理由]	新規登録
住 所	長野県木曽郡木祖村小木曽2977
氏 名	和泉 博